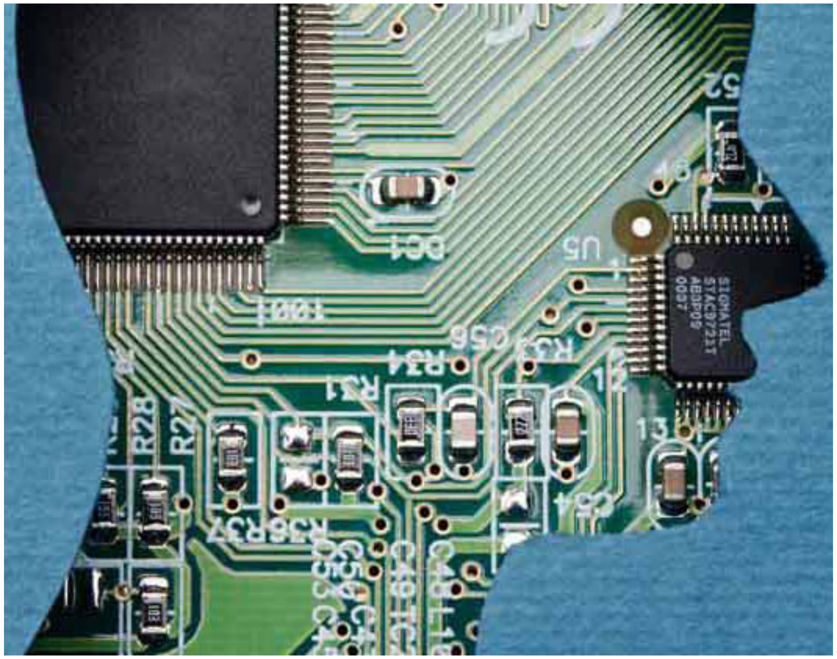
Juno: A Highly Secure, Scalable and Available Key-Value Store With Pluggable Storage Engine

Yaping Shi

Database Engineering @PayPal

May 2020

Agenda



- Why Juno
- Juno Architecture Overview
- Deep Dive: Sharding and Data Redistribution
- Current Status and Next Steps

Why Juno

Secure, consistent, highly scalable and available key-value store providing low (single digit millisecond) latency to meet temporary data store needs of PayPal applications.

Customer Key Pain Points

- High Scalability
- High Availability
- High Security
- Efficiency
- Cloud Enablement

Current Solutions Fall Short

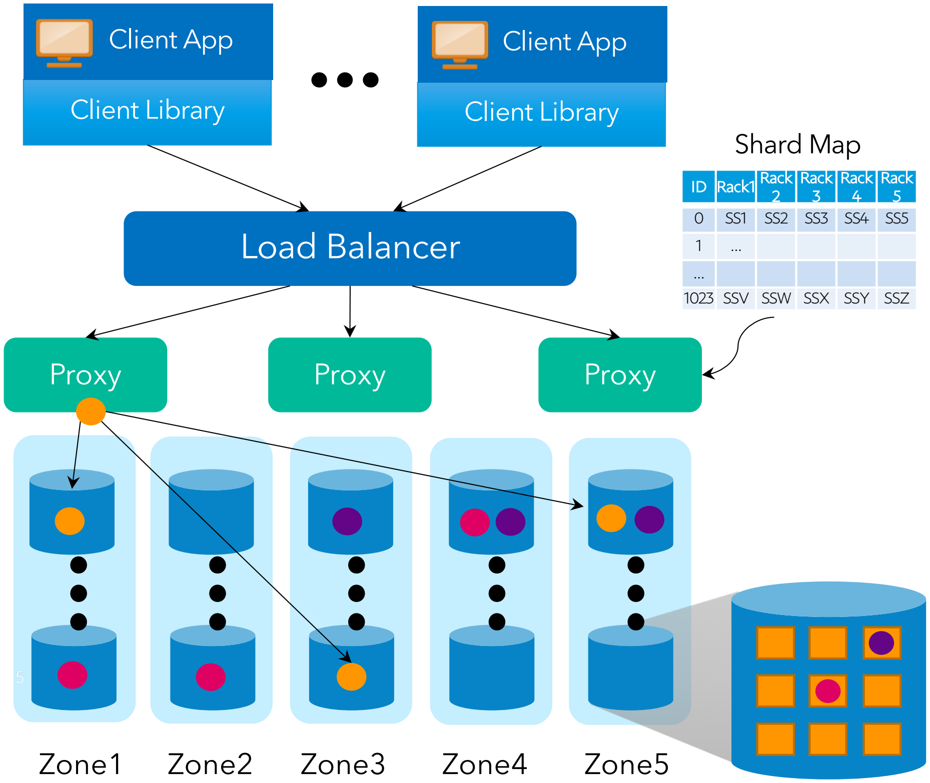
- Couchbase, Aerospike, Cassandra, in-house in-memory K-V Store
- None meets all requirements
- Inefficient to support all, need consolidation

Juno Solves Key Asks

- Connection scalability while preserving throughput and latency
- High availability
- Highly secure

Juno Architecture Overview

Juno Architecture



Shard Map

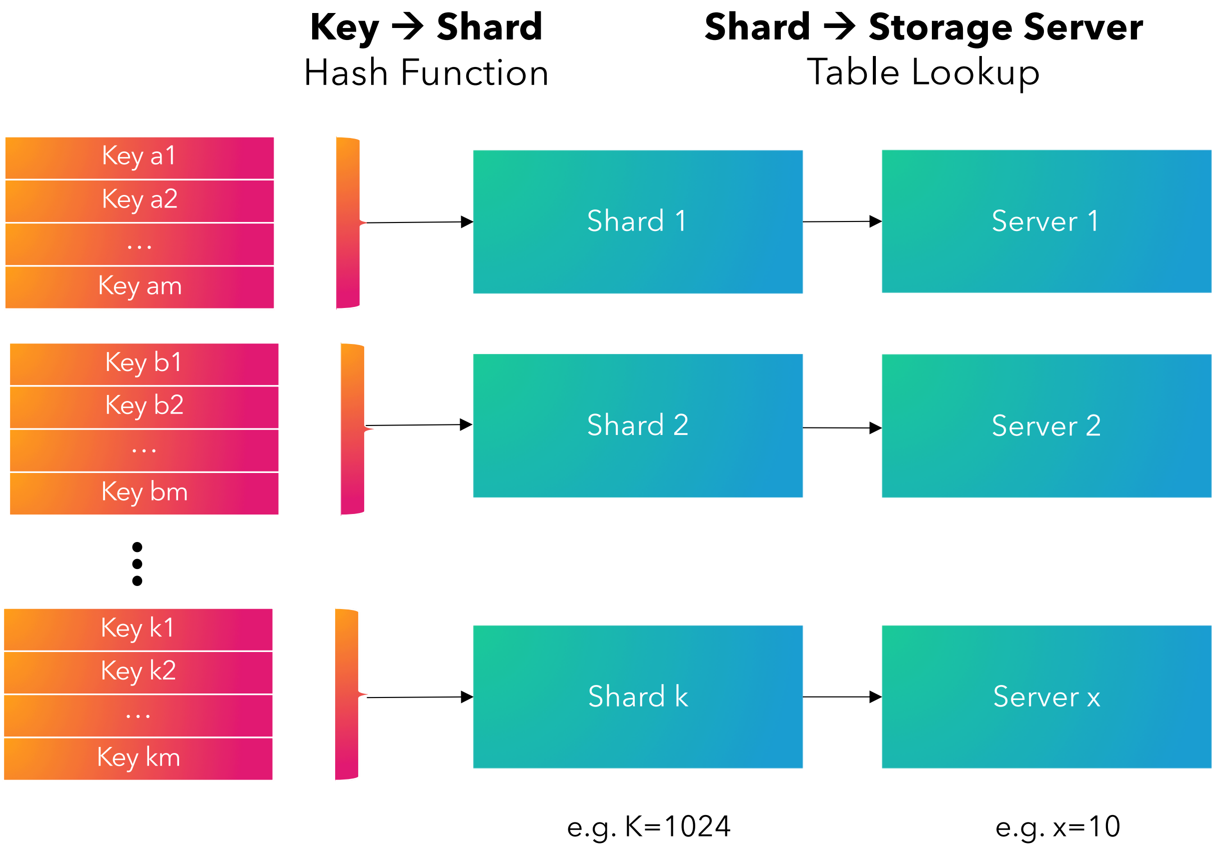
ID	Rack1	Rack 2	Rack 3	Rack 4	Rack 5
0	SS1	SS2	SS3	SS4	SS5
1	...				
...					
1023	SSV	SSW	SSX	SSY	SSZ

- Distributed **horizontally scalable** architecture
  - Thin client, proxy, storage servers
- **Consistent hashing** for incremental scaling
- **Data replication** for fault tolerance and high availability
- **Quorum based** consensus protocol for data consistency
  - W+R > N, W > N/2; ex: W=3, R=3, N=5
  - **Two Phase Commit** for write consistency
- **Highly Secure** with SSL/TLS enabling and at-rest encryption
- **Pluggable Storage Engine** Easy to upgrade to new storage technologies

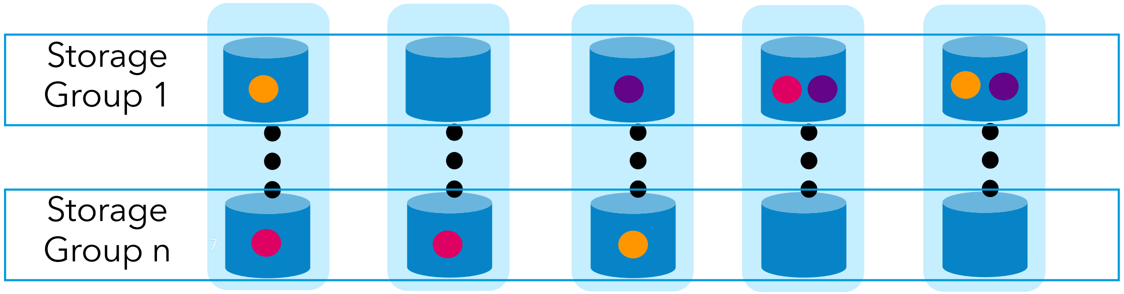
- N=5, W=3 → seven 9s
- N=3, W=2 → five 9s
- N=1, W=1 → three 9s

ATB

vBucket based Consistent Hashing



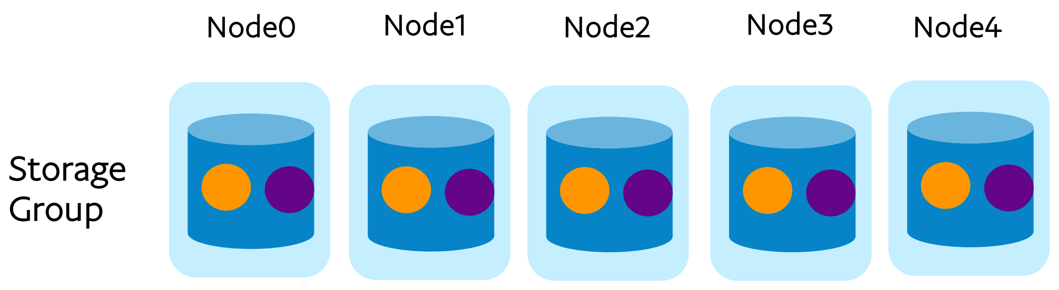
Data Redundancy for Fault Tolerance



	Zone 0	Zone 1	Zone 2	Zone 3	Zone 4
Chunk 0	Primary	Primary	Primary		
Chunk 1		Primary	Primary	Primary	
Chunk 2			Primary	Primary	Primary
Chunk 3	Primary			Primary	Primary
Chunk 4	Primary	Primary			Primary

- **Operation simplicity:** All the zones share the same shard mapping.
- **Redundancy and fault tolerance:** Each shard is replicated to a group of storage nodes located in different zones
- **Quorum based protocol** is used to get consensus on a value in the storage group (3 out of 5): W+R > N, W > N/2; ex: W=3, R=3, N=5
- **Load balance:** We divide the keys in each shard into 5 chunks, determined by hash mod 5. For each chunk, we assign an ordered list of storage nodes.

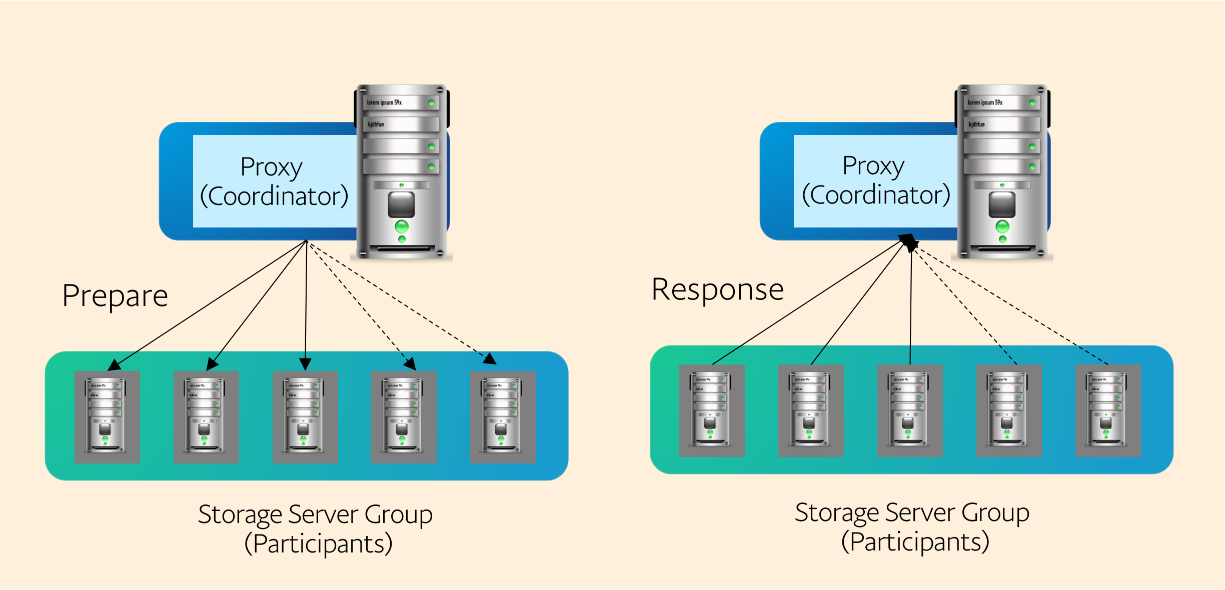
Storage Node Failure Scenario



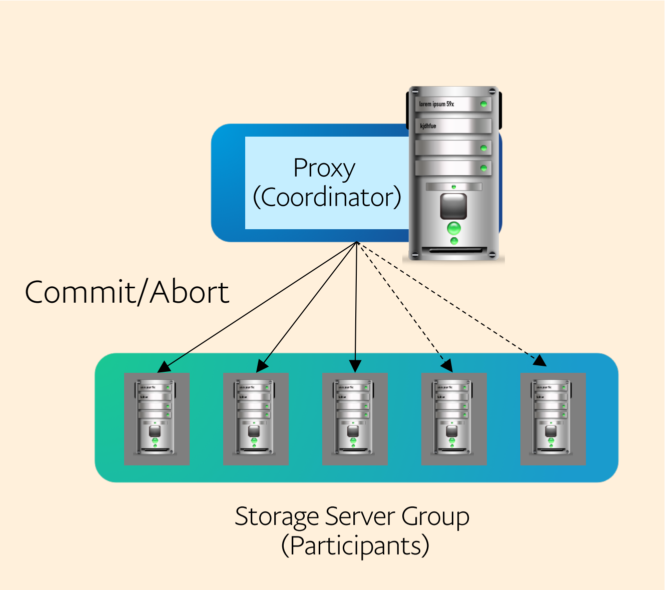
	Zone 0	Zone 1	Zone 2	Zone 3	Zone 4
Chunk 0	Primary	~~Primary~~	Primary	Secondary	
Chunk 1		~~Primary~~	Primary	Primary	Secondary
Chunk 2			Primary	Primary	Primary
Chunk 3	Primary			Primary	Primary
Chunk 4	Primary	~~Primary~~	Secondary		Primary

- Failover is automatic and immediate. No data redistribution needed.
- When zone 1 fails, for chunk 0, the first 3 available nodes in the assigned list will change to (0, 2, 3) from (0, 1, 2).
- We can survive multiple nodes failures as long as there are no more than 2 failures on the same storage group.

Quorum Based Two Phase Commit



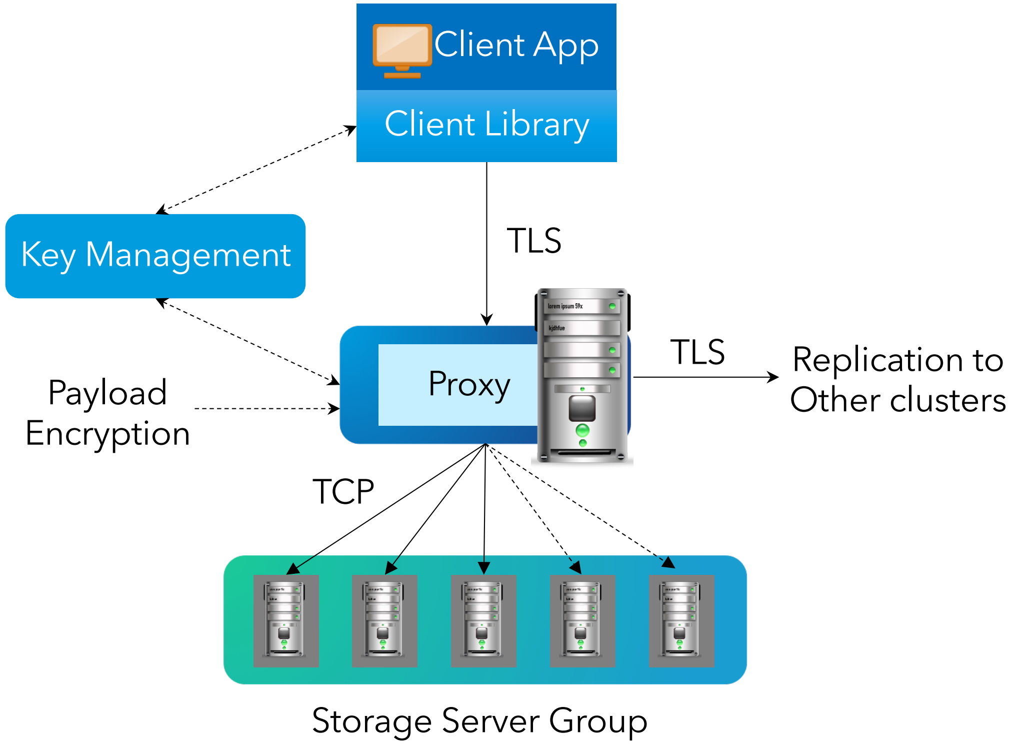
Two-phase commit, phase one



Two-phase commit, phase two

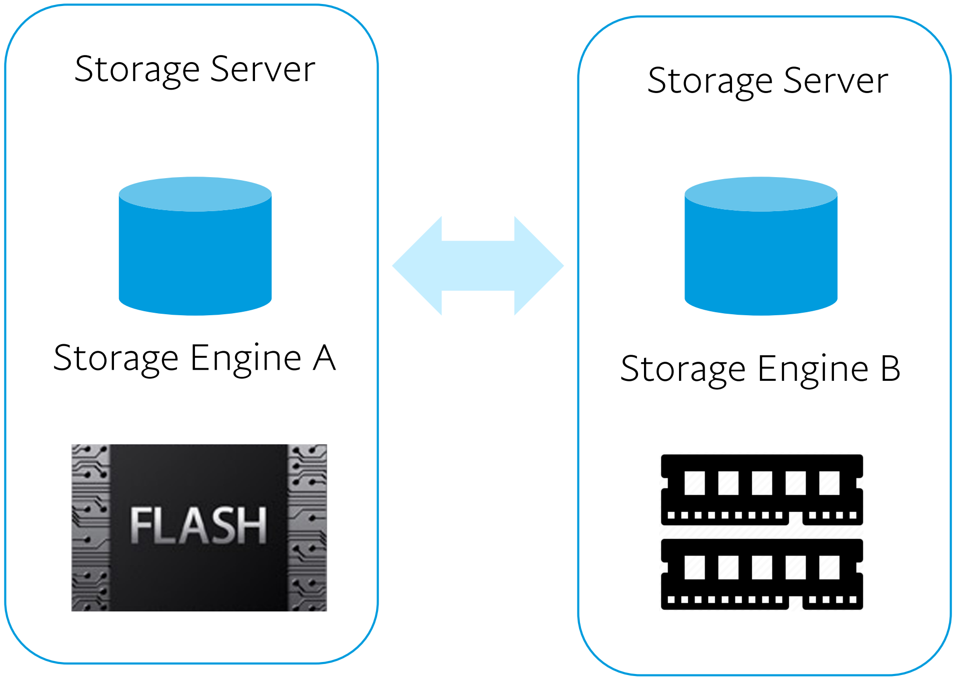
ℹ *Storage server will rollback if commit protocol not complete (due to failure or loss of connection)*

Juno Security



- **TLS:** secure communication with client & replication server
- **Payload encryption** (at client or proxy) for secure storage at rest
- **Key Management module** manages certificate, key distribution and rotation

Pluggable Storage Engine



Storage Server

Storage Engine A

Storage Server

Storage Engine B

Benefits

- Choose underline storage engines based on application needs
- Easy to upgrade to new storage technologies

Supported Engines

- RocksDB as persistent storage engine
- In-house in-memory storage engine

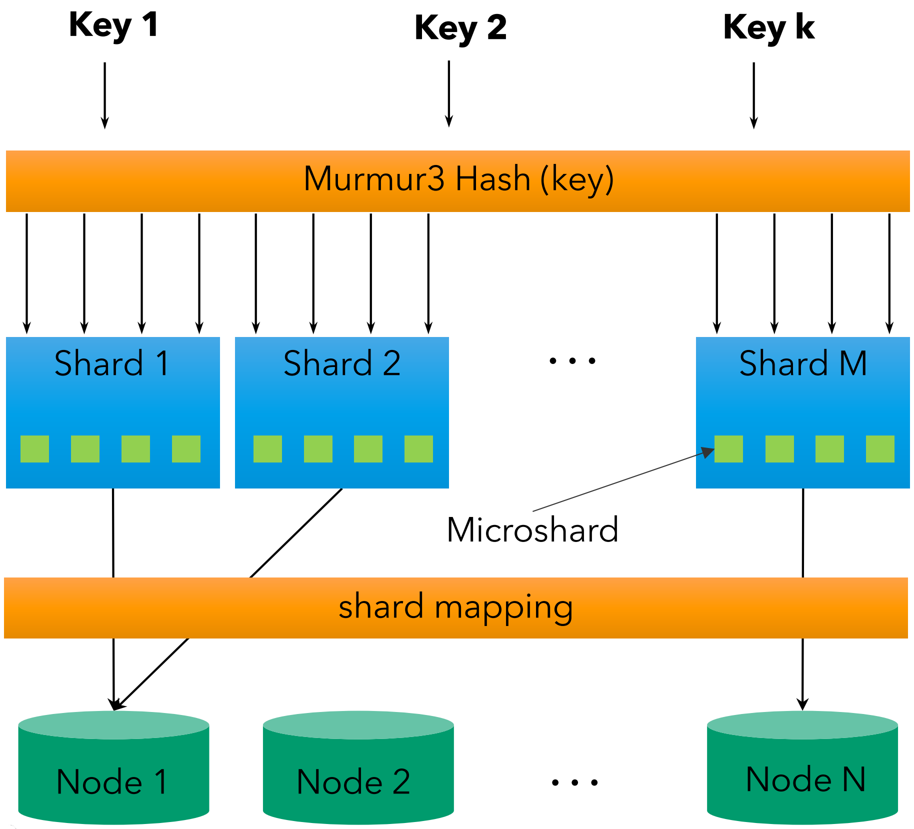
Sharding and Data Redistribution Deep Dive

Data Redistribution: Requirements

When a cluster scales up or down, some shards must be redistributed to different nodes to reflect new cluster topology.

- Minimize the number of shards need to be moved.
- Load balance across each node after redistribution.
- Transparent to client: no downtime and minimizing performance impact.
- Maintain data consistency while performing data redistribution.

Juno Sharding Scheme



- Divide user key space into M logical shards
  - Juno uses Murmur3 for hashing (even key distribution & good performance)
  - Low bits used to calculate Shard ID
  - High bits used to calculate Microshard ID
  - Microshards are only visible within shards and used as a unit for data redistribution
- Assign logical shards to storage nodes using a repeatable and consistent mapping algorithm

Juno Shard Mapping Algorithm

Goal: Minimize shard moves during redistribution with a consistent mapping

Solution: Start with one node, then add nodes one at a time using the following algorithm

1. Rank nodes by number of shards in descending order (tiebreak using node id, in descending order)
2. Move shard with highest shard id from the highest ranked node to the new node
3. Repeat steps 1 & 2 until the new node has enough shards (at least average)

num nodes	Shard id (total 32 shards)																														
/zone	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2
4	0	0	0	0	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	1	1	1	3	3	3	2	2	2	3	3
5	0	0	0	0	0	0	0	4	3	3	3	2	2	2	2	2	1	1	1	1	1	1	1	4	3	3	3	2	4	4	4	4
6	0	0	0	0	0	0	5	4	3	3	3	2	2	2	2	2	1	1	1	1	1	1	5	4	3	3	5	5	4	4	4	5
7	0	0	0	0	0	6	5	4	3	3	3	2	2	2	2	2	1	1	1	1	1	6	5	4	3	3	5	5	4	4	6	6
8	0	0	0	0	7	6	5	4	3	3	3	2	2	2	2	7	1	1	1	1	7	6	5	4	3	7	5	5	4	4	6	6

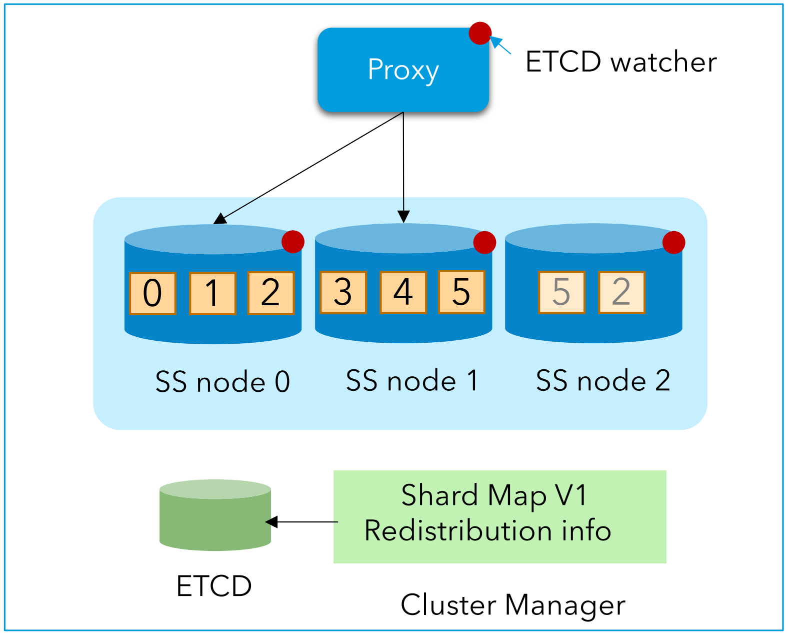
Data Redistribution: Execution Plan

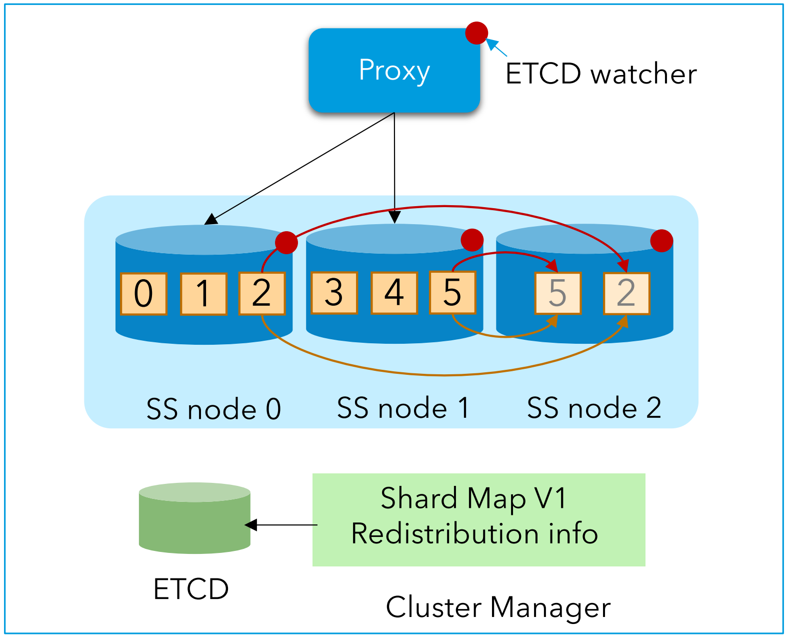
Execution Plan to scale cluster from 2 nodes to 4 nodes

# nodes per zone	Shard Id (Total 32 Shards)																														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2 nodes	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4 nodes	0	0	0	0	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	1	1	1	3	3	3	2	2	2	3	3

Shards	**Move**
Shard 8, 9, 10	Node 0 → Node 3
Shard 11, 12, 13, 14, 15	Node 0 → Node 2
Shard 27, 28, 29	Node 1 → Node 2
Shard 24, 25, 26, 30, 31	Node 1 → Node 3

Data Redistribution: Cluster Scale Up





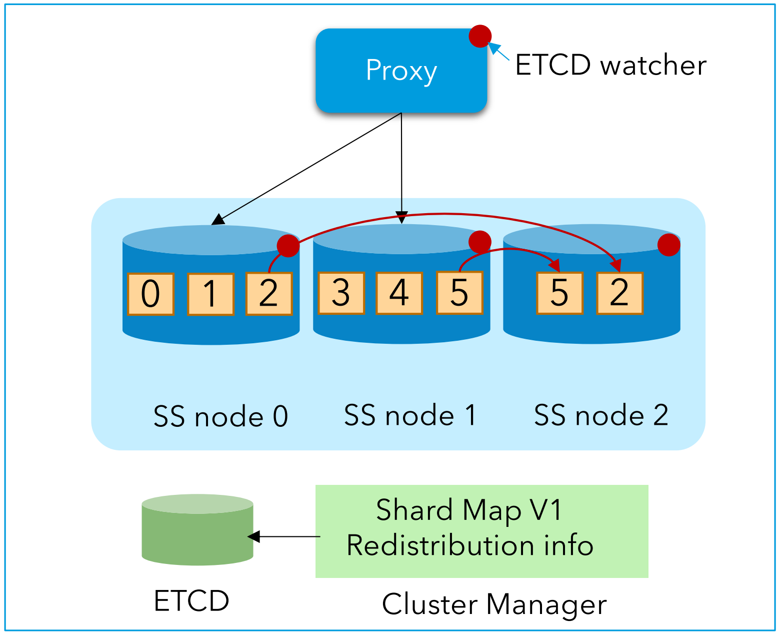
1. Preparation

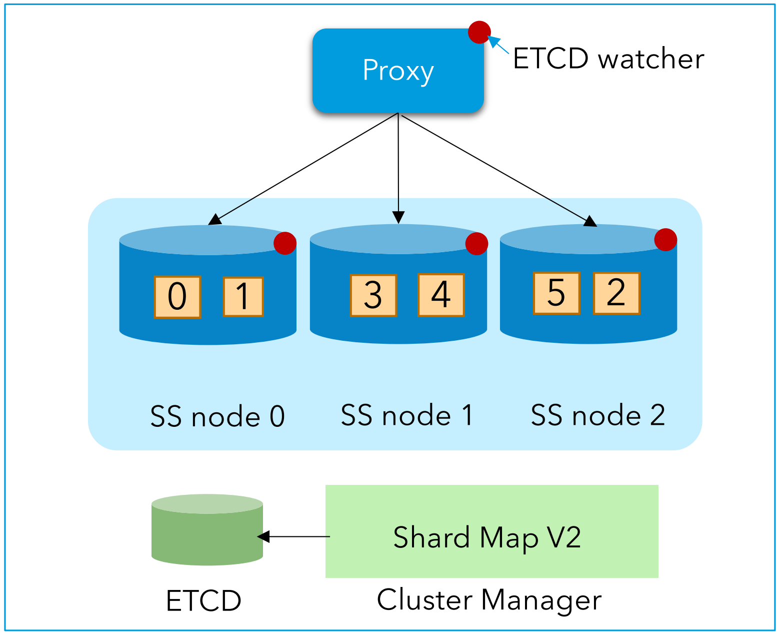
- Prepare new node
- Insert execution plan into ETCD
  - Shard 2, node 0 -> node 2,
  - Shard 5, node 1 -> node 2

2. Trigger redistribution

- Replicate real time requests (red arrows)
- Transfer snapshot (orange arrows)

Data Redistribution: Cluster Scale Up II





3. Snapshot transfer complete

- Validate data

4. Finish

- Update shard Map
- Stop real time request forwarding

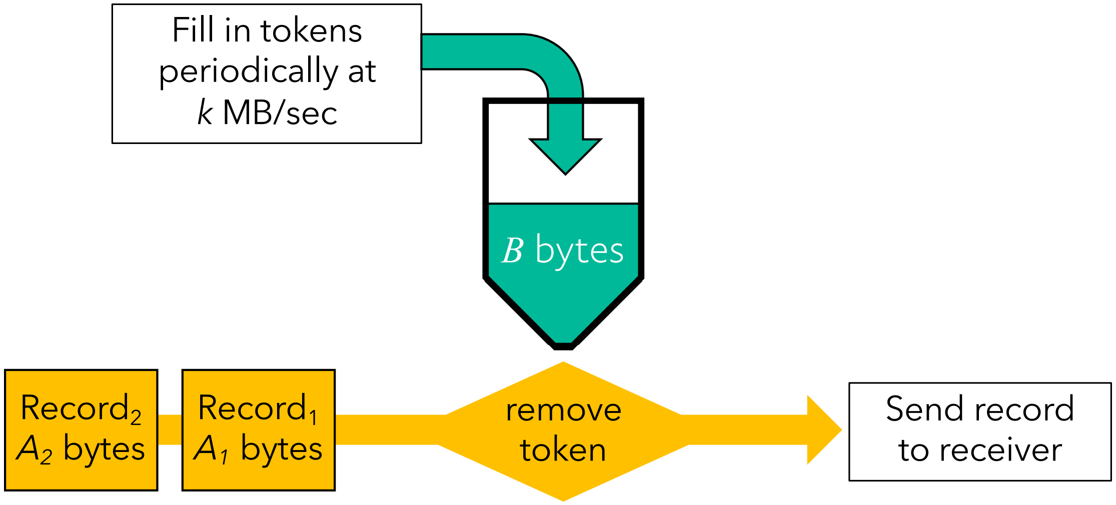
Data Redistribution: Rate Limiter

Problem

- Live incident (Couchbase): one node down, data redistribution for secondary shards caused system meltdown
- Without rate limiter, receiver can be overwhelmed

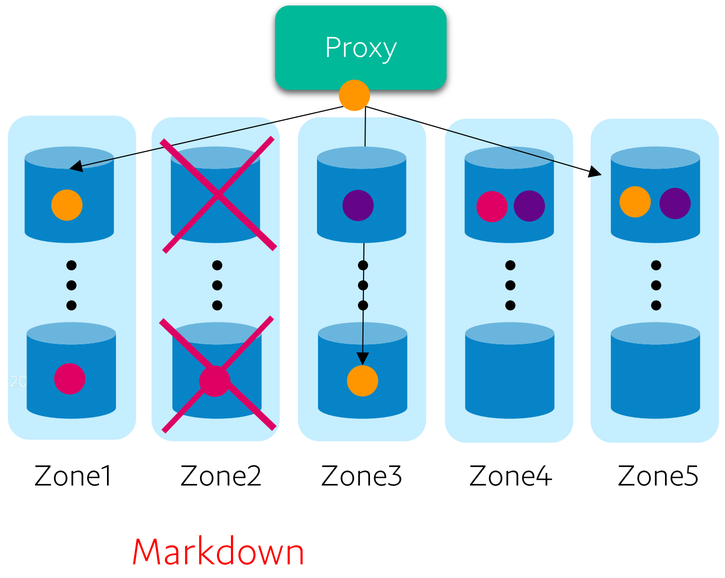
Solution

- Token bucket-based rate limiter
- Use SSD IOPS, network throughput, number of SS instances per box, and real-time traffic estimate to **set the rate** *k* MB/second



- For every record of *A* bytes, remove *A* bytes worth of token from bucket.
- If not enough token, wait.

Data Redistribution: Zone Markdown



- When we expand cluster zone by zone, Juno provides an **option to markdown a zone to not take real-time traffic**
- **Real-time data will be processed by other storage nodes** in the group. Zone 2 resources are dedicated to redistribution.
- Other storage nodes will take 25% more real time traffic, and so there is **no significant performance impact during redistribution**.

Current Status and Next Steps

Current Status

- Rolled out in PayPal production since Nov 2019
- Deployed 18 live clusters across the regions

Next Steps

- Migrate most of existing and future key-value use cases to Juno
- Cloud enablement
- Open Source
- New Storage Technologies

Q&A